Universidad Zaragoza
1542

# Trabajo Fin de Grado

Título del trabajo: Desarrollo de una aplicación Android para la co-evaluación de exposición de trabajos.

English title: Android app development for co-evaluating students oral presentations.


Autor/es

Miguel Delgado Corral

Director/es

Dr. José Luis Salazar Riaño


Titulación del autor

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación


ESCUELA DE INGENIERÍA Y ARQUITECTURA

Año 2020

# Resumen

El objetivo fundamental de este trabajo, es desarrollar una aplicación Android funcional para que los alumnos de la universidad puedan llevar a cabo la co-evaluación de las exposiciones orales de sus compañeros.

Para ello, en primer lugar, se ha creado mediante la plataforma Firebase de Google una base de datos. Esta base de datos cuenta con cuatro tipos de colecciones (alumnos, profesores, asignaturas y presentaciones) y una sub-colección que se ubica dentro de la colección presentaciones (calificaciones).

Firebase proporciona además un sistema de autenticación de usuarios que se ha usado para poder distinguir si el usuario que entra a la aplicación es un alumno o un profesor.

La co-evaluación se lleva a cabo de la siguiente forma: Un profesor crea un documento presentación en la base de datos y los alumnos crean documentos calificación en dicha presentación.

Para garantizar la presencia de los alumnos en la presentación se ha implementado un sistema que, mediante el envío de una contraseña en los paquetes de advertising BLE, verifica la presencia de los alumnos en la presentación, de esta forma se evita que un alumno malicioso pueda enviar calificaciones sin estar siquiera presente en la presentación.

# Índice

# 1. Introducción

Los Smartphone han supuesto una revolución de nuestro estilo de vida a lo largo de la última década. La capacidad de tener una conexión a internet en cualquier momento mediante un dispositivo portátil otorga una infinidad de posibilidades, desde simplemente leer el periódico a controlar el ritmo cardiaco y avisar a emergencias si hay alguna complicación.

El número de usuarios de Smartphone ha aumentado considerablemente en los últimos años y hoy en día prácticamente cualquier persona de entre 15 y 55 años en España dispone de uno.

En cuanto al sistema operativo, hay dos grandes competidores: Android y Apple. Aún así, Android ocupa una cuota de mercado a nivel mundial de casi el 75% en contraste con el 23% de IOS. La comparación entre Android e IOS se explica en mayor profundidad en la *sección 3.2*.

Como se ha mencionado anteriormente, prácticamente cualquier persona dispone de un Smartphone, la mayoría de ellos usan el sistema operativo android, y éstos nos ofrecen una gran cantidad de posibilidades. Una de esas puede ser ayudarnos a que los alumnos puedan realizar la co-evaluación del trabajo de sus compañeros.

La evaluación es un acto que, tradicionalmente, ha sido asociado exclusivamente al profesorado, sin embargo existen numerosos estudios que señalan los beneficios de una participación activa del propio alumnado en la misma, la llamada co-evaluación o evaluación entre pares.

Dichos estudios indican que los alumnos adquieren competencias multidisciplinares como la abstracción, el desarrollo de argumentos, y la capacidad de describir, evaluar, criticar, analizar y revisar.

Este TFG se ha desarrollado como una prueba de concepto de una aplicación móvil android para la co-evaluación de las presentaciones orales que se llevan a cabo durante la trayectoria académica.

Mediante el uso de esta aplicación, los alumnos podrán evaluar las presentaciones llevadas a cabo por sus compañeros, mientras que el profesor tendrá control completo sobre las mismas, pudiendo ver en tiempo real las calificaciones otorgadas por los alumnos y decidiendo cuando empieza y cuando finaliza el proceso de calificación.

Mediante la tecnología Firebase de Google se ha creado una base de datos en la que están guardados una lista con los alumnos y profesores del centro perfectamente autenticados, con las asignaturas y con las presentaciones que se vayan creando.

Para verificar la presencia de los alumnos en la presentación que van a evaluar, se ha usado la comunicación mediante el envío de paquetes Bluetooth Low Energy (BLE).
La aplicación tiene dos roles: profesor y alumno. El profesor es el encargado de crear las presentaciones en la base de datos, así como de generar una contraseña única para

cada presentación y enviarla vía BLE en un beacon broadcast. El alumno, para poder votar, debe conocer la contraseña que ha recogido del beacon BLE.

El funcionamiento de la aplicación es el siguiente: Cuando un alumno realiza una presentación, el profesor desde su Smartphone crea una entrada en la base de datos que corresponde a la presentación, esta entrada llevará asociada la asignatura a la cual está asociada dicha presentación.

Una vez creada la "presentación", automáticamente su Smartphone comienza a enviar paquetes BLE en modo broadcast con una contraseña necesaria para poder calificar la presentación. Debido al limitado rango del bluetooth (aproximadamente 10 metros), esto permite verificar la asistencia de los alumnos a la presentación. Mientras envía paquetes, el profesor ve en su pantalla las calificaciones en tiempo real que los alumnos van subiendo a la base de datos.

Cuando el profesor se asegure de que todos los alumnos presentes han votado, no tiene más que apretar un botón en la pantalla de su Smartphone para cerrar la presentación e impedir nuevas votaciones.

Desde el punto de vista del alumno la aplicación es incluso más intuitiva, simplemente debe pulsar un botón para escanear en busca de presentaciones y cuando la encuentre se conectará automáticamente. Una vez conectado a la presentación simplemente deberá introducir la calificación que crea conveniente y pulsar un botón para subirla a la base de datos.

En las figuras a continuación se muestra un diagrama de flujo de la información:

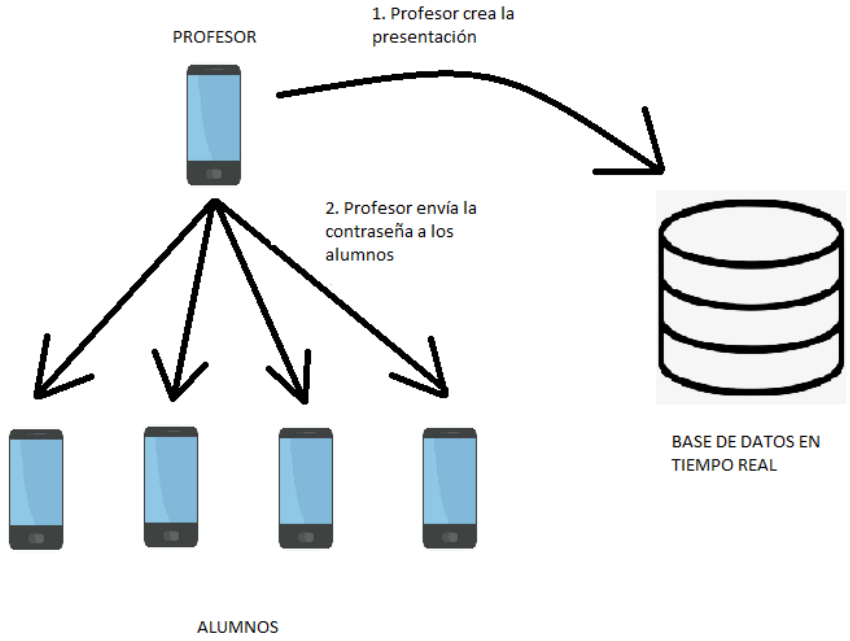


*Figura 1.1. Escenario co-evaluación (1).*

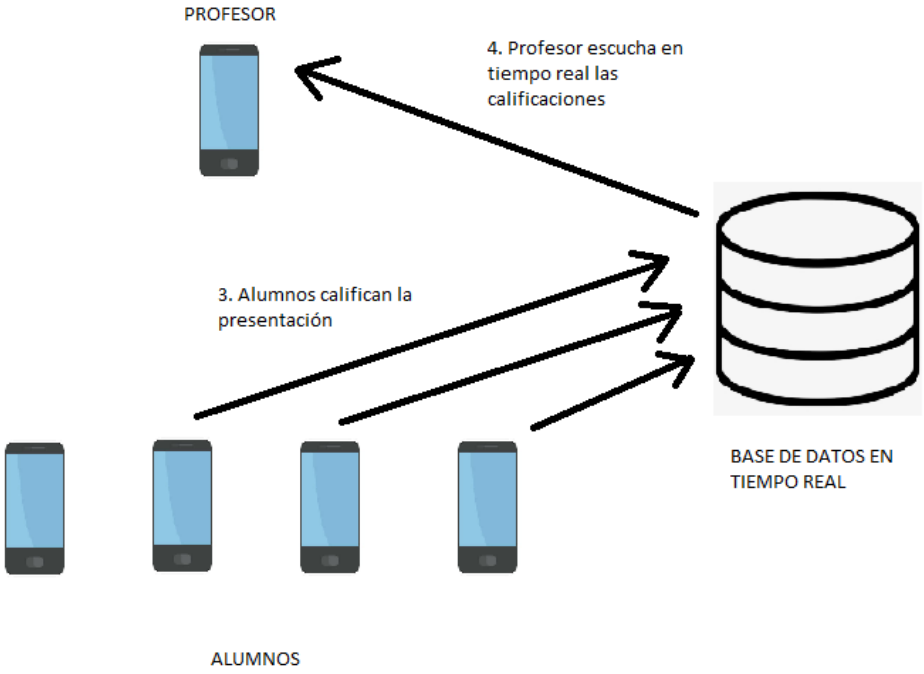


*Figura 1.2. Escenario co-evaluación (2).*

# 2. Base de datos

## 2.1. Definición de requisitos

La base de datos debe cumplir diversas funciones:

- En primer lugar debe almacenar una lista de los alumnos y los profesores que van a participar en el proceso de co-evaluación. Esto se relaciona directamente con la autenticación de usuarios en la aplicación, ya que se buscará en la base de datos para decidir si el usuario que se está autenticando es un alumno o un profesor.

- Debe contener también una lista con todas las asignaturas que se imparten, y relacionar de alguna forma esta lista con las listas de alumnos y profesores previamente mencionadas, para así poder saber que alumnos están matriculados en cierta asignatura y también qué profesores la imparten.

- Es la encargada de almacenar tanto las presentaciones que crean los profesores como las calificaciones que los alumnos otorgan a sus compañeros.

- La base de datos debe permitir la sincronización de datos entre los distintos clientes en tiempo real, para que el profesor pueda visualizar en su smartphone las calificaciones que los alumnos van poniendo a las presentaciones de sus compañeros.

## 2.2. Análisis

El primer punto crítico a la hora de empezar a crear desde cero una base de datos, ha sido tomar la decisión entre realizar la programación completa de la base de datos o usar una API de algún proveedor que nos proporcione un acceso sencillo a una base de datos sin necesidad de tener que controlar el hardware y el software del servidor, es decir, que se encargue de controlar el back-end de la aplicación. Debido a la limitación de recursos para crear un servidor dedicado para la aplicación, se ha tomado la decisión de usar una API.

Una vez tomada esta decisión, se ha realizado una búsqueda entre los diferentes proveedores que ofrecen una base de datos en tiempo real de manera gratuita, encontrándose dos opciones principales: Firebase de Google y Amazon Web Services (AWS).

Otras opciones interesantes son Back4App o Kinvey pero fueron descartadas rápidamente ya que, al no estar respaldadas por grandes compañías cómo Google o Amazon, son menos populares y por lo tanto es más difícil encontrar información acerca de cómo trabajar con ellas e ir solventado los problemas que van surgiendo a la hora de interactuar con la base de datos. En cambio existen infinidad de hilos en páginas de consulta para programadores como stackoverflow acerca de Firebase y AWS y adicionalmente la información que dan en sus guías acerca de cómo empezar a utilizar sus servicios es mucho más completa.

### *2.2.1. Comparación entre Firebase y AWS*

Firebase es una plataforma para facilitar el desarrollo de aplicaciones web y aplicaciones móviles comprada por Google en 2014.

Esta plataforma cuenta con varias funcionalidades para minimizar el tiempo de optimización y desarrollo de las aplicaciones siendo las más destacadas las siguientes:

- **Firebase Cloud Messaging**: Plataforma para el envío de mensajes y notificaciones.
- **Firebase Auth**: Servicio para autenticar a los usuarios usando código solamente desde el lado del cliente. Se puede usar para la autenticación de proveedores de inicio de sesión como Facebook o Google y también permite la autenticación con e-mail y contraseña.
- **Firebase Storage**: Provee la capacidad de realizar cargas y descargas de archivos. Está respaldado por Google Cloud Storage.
- **Realtime Database**: Proporciona una base de datos organizada en forma de árbol que almacena y sincroniza los datos de todos los clientes conectados en tiempo real.
- **Cloud Firestore**: Ofrece una base de datos NoSQL con actualizaciones en tiempo real y ajuste de escala automático. Es el sucesor de Realtime Database, en vez de la estructura en forma de árbol, permite el uso de documentos anidados y campos.

Dos de las funcionalidades anteriores son especialmente relevantes para la aplicación propuesta: Firebase Auth para la autenticación de usuarios y Cloud Firestore para la base de datos en tiempo real.

Firebase cuenta con muchas otras funcionalidades, no sólo para el desarrollo de aplicaciones web y móviles, sino también para gestionar el posible crecimiento del número de usuarios de la aplicación e incluso para la monetización.

Por último cabe destacar Firebase Analytics, que provee información sobre el uso interno de la aplicación.


*Figura 2.1. Funcionalidades de Firebase.*

Por otro lado, AWS provee de una cantidad de funcionalidades muy superior a Firebase (tantos que es imposible enumerarlos todos) pero destacaremos las más importantes:

- **AWS Lambda**: Permite ejecutar código sin aprovisionar ni administrar servidores. Se puede utilizar para realizar labores de autenticación de usuarios.
- **Amazon DynamoDB**: Proporciona un servicio de base de datos NoSQL rápido (milisegundos) y flexible para cualquier escala.
- **Amazon RDS**: Proporciona un servicio de bases de datos relacionales administrado para MySQL, PostgreSQL, MariaDB, Oracle BYOL o SQL Server.
- **Amazon SageMaker**: Servicio completamente administrado que brinda a los desarrolladores la capacidad de crear, entrenar e implementar modelos de aprendizaje automático de forma rápida
- **Amazon EC2**: Es un servicio web que proporciona capacidad informática en la nube segura y de tamaño modificable, simplificando el uso de la informática en la nube a escala web para los desarrolladores.

Para la aplicación propuesta son especialmente importantes AWS Lambda que permite realizar autenticación de usuarios y Amazon DynamoDB para la base de datos en tiempo real.

Realizando una comparación entre los servicios ofrecidos por Firebase y los que ofrece AWS, queda claro que los de AWS son más extensos, pero la curva de aprendizaje para comenzar a usarlos es bastante mayor que en Firebase, que es mucho más intuitivo.

AWS puede ser más adecuado para proyectos de una gran extensión y complejidad mientras que para proyectos de una complejidad media como éste, Firebase parece más adecuado, ya que proporciona todas las funcionalidades necesarias y es sensiblemente más sencillo de utilizar.

Adicionalmente, el servicio AWS Lambda es de pago, por lo que habría que pagar por cada vez que un usuario se autenticase mientras que Firebase proporciona FirebaseAuth de forma gratuita.

Por los motivos que se acaban de exponer, se ha decidido finalmente utilizar la plataforma de Firebase para llevar a cabo la autenticación de usuarios y la base de datos.

## 2.3. Diseño

En esta sección se describe el diseño de la base de datos mediante la tecnología Firebase de Google. La base de datos ha sido desarrollada como una prueba de concepto, con un número limitado tanto de alumnos como de profesores y de asignaturas, pero dado que Firebase es escalable es factible la repetibilidad con bases de datos más extensas.

### 2.3.1. Diseño de la base de datos

La base de datos de Cloud Firestore, está organizada mediante colecciones, documentos y campos.

Todos los documentos deben estar almacenados dentro de alguna colección y a su vez cada documento puede contener subcolecciones y objetos anidados que pueden contener campos (que pueden ser: String, Número, Booleano, Mapa con otros campos anidados, Array, Timestamp con fecha y hora, Geolocalización con latitud y longitud o una referencia a otro documento de la base de datos).

Cada documento está identificado con un nombre (llamado ID de aquí en adelante), y representa la unidad de almacenamiento de Cloud Firestore. Al crear un documento, Cloud Firestore asigna de forma automática un ID de 20 bytes al mismo a no ser que se especifique lo contrario y se le asigne un ID manualmente.

Por otro lado, las colecciones se utilizan para almacenar los documentos, y no pueden contener campos sin procesar ni tampoco otras colecciones.

Las colecciones y documentos se crean de manera implícita en Cloud Firestore, es decir, al intentar escribir en una colección o un documento que no existen, Cloud Firestore los crea de manera automática.

La base de datos que se ha diseñado para este proyecto, consta de cuatro colecciones principales: Alumnos, Asignaturas, Profesores y Presentaciones, que se describen a continuación

- **Alumnos**: Contiene un documento (cuyo nombre será llamado a partir de ahora ID alumno) por cada alumno que ha sido creado, el cual a su vez contiene un campo de tipo String con el nombre del alumno. Un ejemplo de esta estructura de datos, se muestra en la *figura 2.2*. Para esta prueba de concepto se han creado un total de cuatro alumnos.

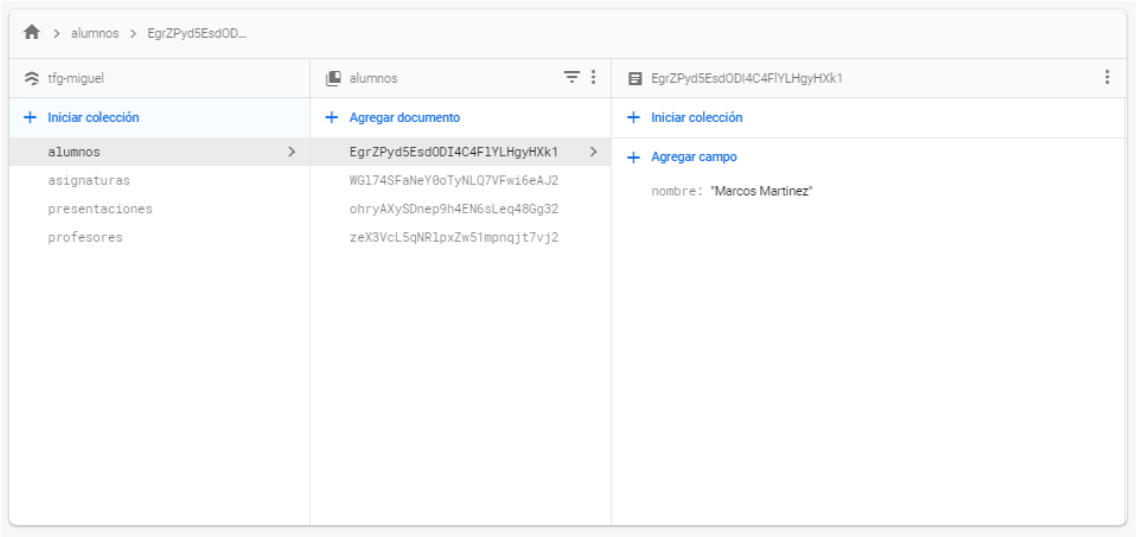


*Figura 2.2. Colección alumnos.*

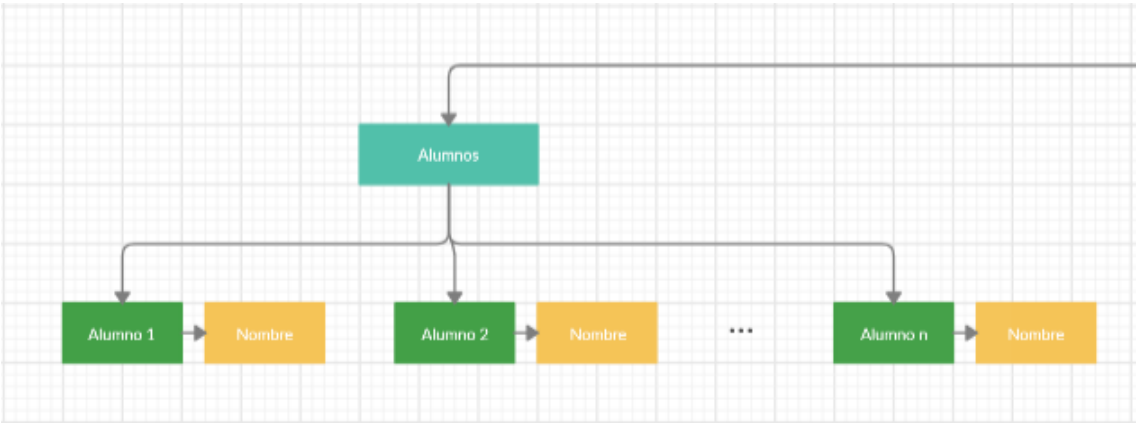


*Figura 2.3. Diagrama de árbol de la colección alumnos.*

- **Profesores**: Contiene un documento (cuyo nombre será llamado a partir de ahora ID profesor) por cada profesor que ha sido creado, el cual a su vez contiene un campo de tipo String con el nombre del alumno. Un ejemplo de esta estructura de datos, se muestra en la *figura 2.4.* Para esta prueba de concepto se han creado un total de tres profesores.

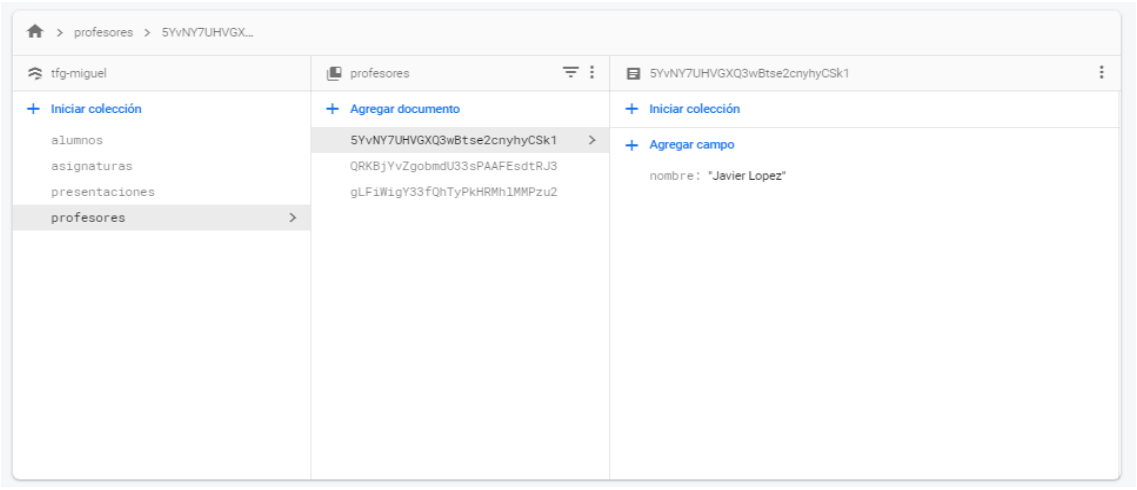


*Figura 2.4. Colección profesores.*

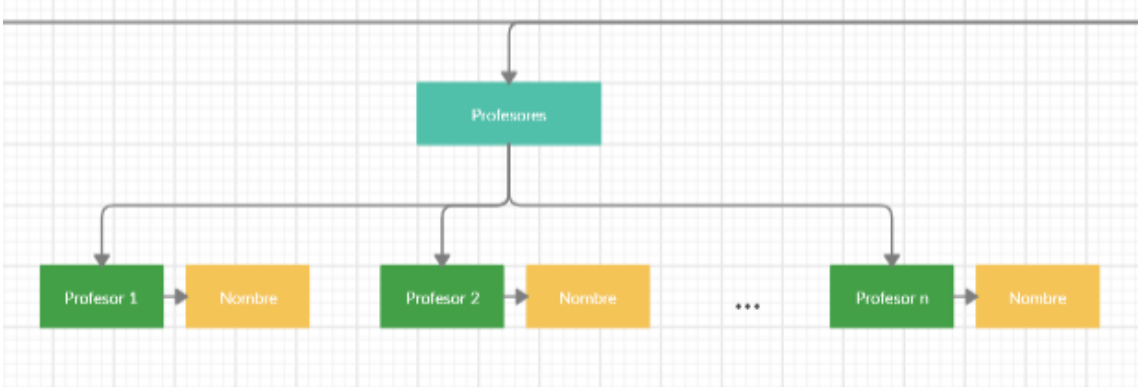


*Figura 2.5. Diagrama de árbol de la colección profesores.*

- **Asignaturas**: Contiene un documento (cuyo nombre será llamado a partir de ahora ID asignatura) por cada asignatura que ha sido creada. El ID de las asignaturas ha sido creado manualmente (por código) para que sea de 4 bytes, el motivo detrás de esta decisión de diseño está explicado en la *sección 3.5.4*. Cada documento asignatura consta de tres campos, uno de tipo String con el nombre de la asignatura, otro de tipo array con el ID de todos los alumnos matriculados en la asignatura uno en cada posición y un tercero de tipo array con el ID de los profesores que imparten la asignatura uno en cada posición del array. Un ejemplo de esta estructura de datos, se muestra en la *figura 2.6.* Para esta prueba de concepto se han creado un total de cinco asignaturas.

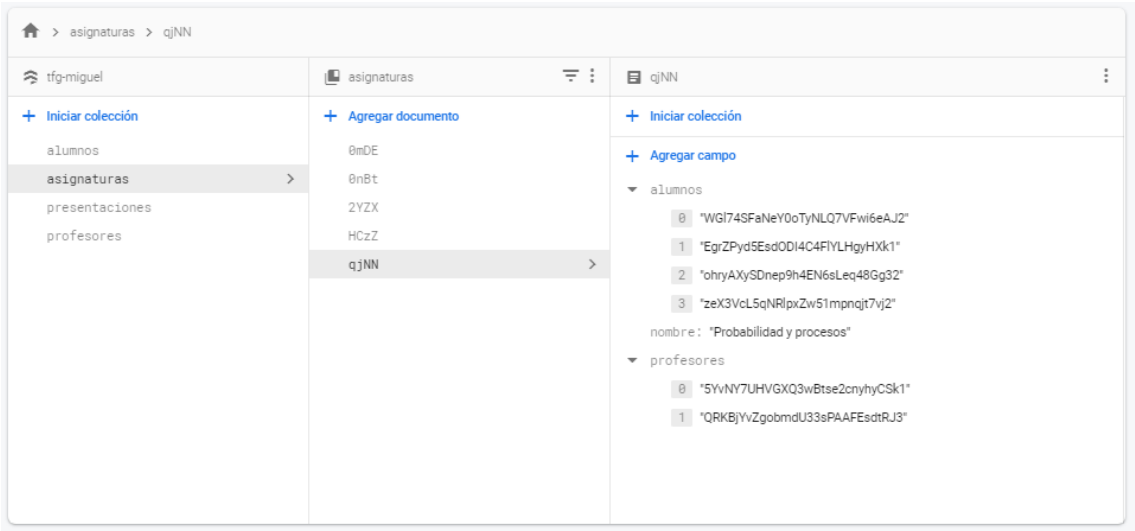


*Figura 2.6. Colección asignaturas.*

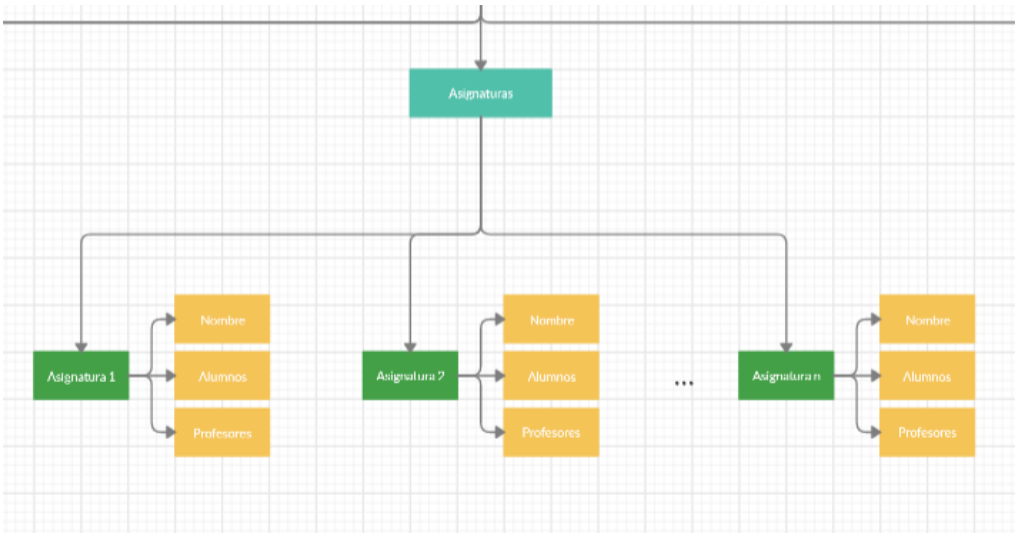


*Figura 2.7. Diagrama de árbol de la colección asignaturas.*

- **Presentaciones**: Contiene un documento por cada presentación que se crea (cuyo nombre será llamado a partir de ahora ID presentación). Cada documento consta de cinco campos, cuatro de ellos de tipo String que son: id_asignatura que contiene el ID de la asignatura de la que se trata la presentación en cuestión, nombre asignatura con el nombre de la asignatura, nombre presentación con el nombre de la presentación y owner con el ID del profesor que ha creado la presentación. Además consta de un último campo de tipo booleano llamado isFinished que es true cuando la presentación ha acabado y false mientras está activa. Un ejemplo de esta estructura de datos, se muestra en la *figura 2.8.*

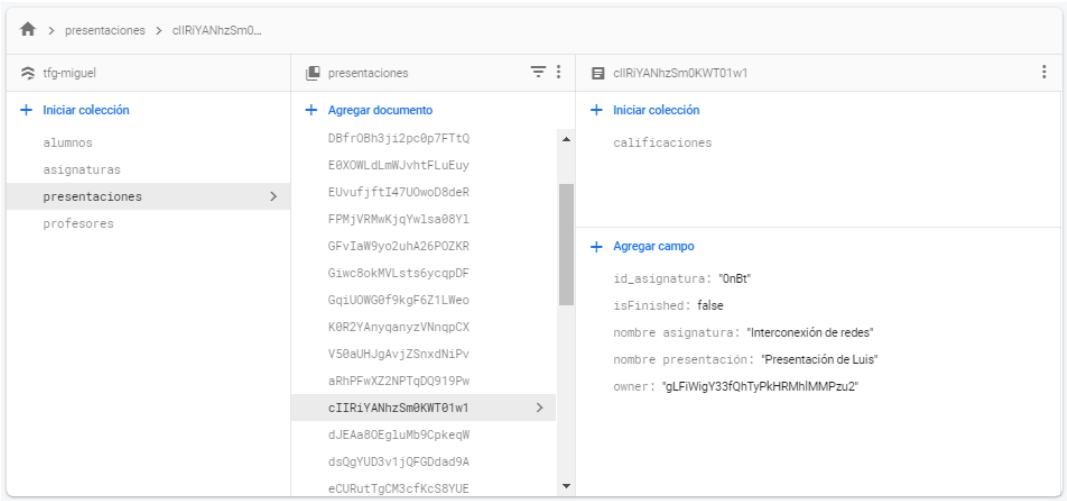


*Figura 2.8. Colección presentaciones.*

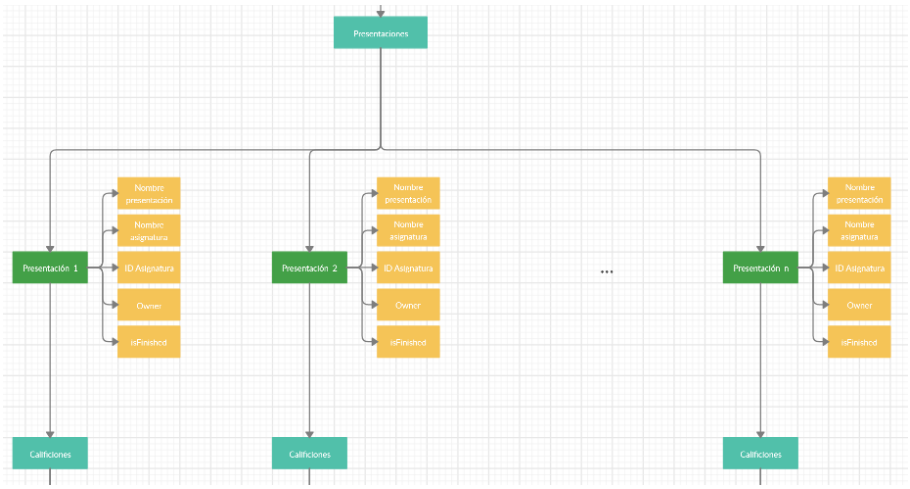


*Figura 2.9. Diagrama de árbol de la colección presentaciones.*

Cada documento presentación contiene una sub-colección llamada Calificaciones que se crea automáticamente cuando un alumno envía la primera calificación. Cada nota enviada por un alumno a la base de datos genera un documento (con ID documento igual a su propio ID alumno) dentro de dicha sub-colección que a su vez contiene dos campos: uno de tipo Número llamado calificación, que contiene la nota enviada por el alumno, y otro de tipo String llamado nombre_alumno que contiene el nombre del alumno que ha enviado la calificación. Un ejemplo de esta estructura de datos, se muestra en la *figura 2.10.*

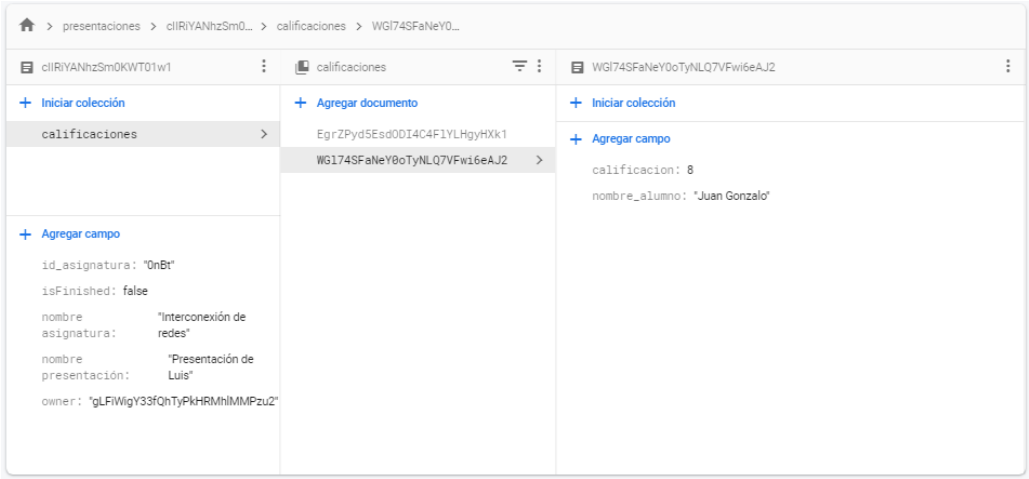


*Figura 2.10. Sub-colección calificaciones.*

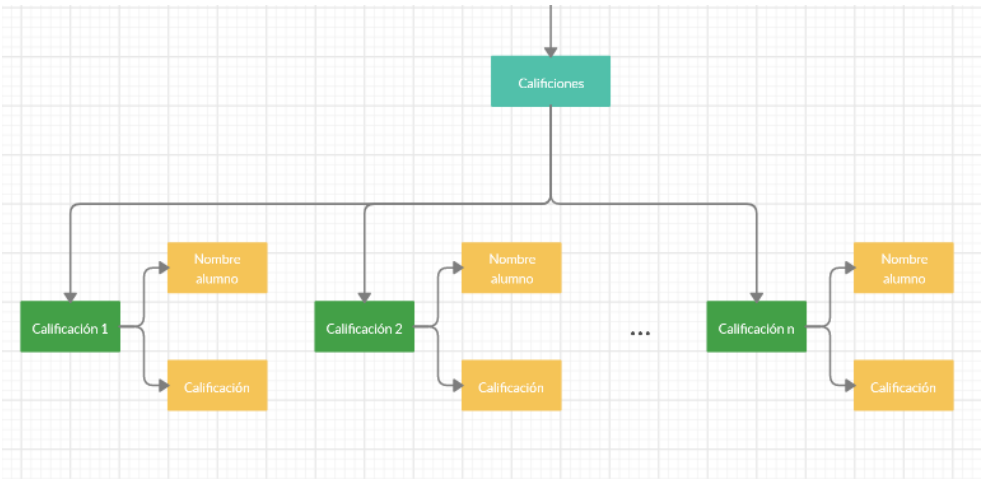


*Figura 2.11. Diagrama de árbol de la sub-colección calificaciones.*

Los valores que se asignan a los campos tanto de la colección presentaciones como de sus datos anidados están explicados más en detalles en la *sección 3.6*.

### *2.3.2. Diseño de las reglas de seguridad*

En Cloud Firestore, es necesario programar las reglas de seguridad de la base de datos para controlar el acceso de lectura y escritura a los documentos y colecciones de la base de datos. Para que un usuario tenga acceso de lectura (operación read) o de escritura (operación write) a un documento o colección, hay que otorgárselo específicamente en las reglas de seguridad, de lo contrario la operación será denegada.

La operación write puede desglosarse en 3 sub-operaciones, a saber: crear (operación create), borrar (operación delete) y actualizar (operación update).

Las reglas definidas para una colección, no se aplican a las sub-colecciones que estén contenidas en documentos de esa colección, por lo que hay que definir reglas específicas también para las sub-colecciones.

Para el diseño de las reglas de seguridad de la base de datos, se han considerado los tres tipos de usuarios que pueden intentar acceder a ella: usuarios sin autenticar, usuarios autenticados como alumnos y usuarios autenticados como profesores.

Para las colecciones alumnos, profesores y asignaturas se han diseñado las mismas reglas de seguridad: nadie (salvo el administrador) puede escribir en ellas y en los documentos contenidos ellas y cualquier usuario autenticado puede leerlas. Estas reglas se han diseñado de esta forma para que ningún usuario malicioso pueda modificar las listas de alumnos, profesores o asignaturas, pero sin embargo cualquier usuario autenticado pueda consultarlas.

En cuanto a las reglas de seguridad de la colección presentaciones hay que tener en cuenta que las reglas no se propagan a las sub-colecciones, por lo que se han definido reglas distintas para la colección presentaciones y para la sub-colección calificaciones dentro de los documentos presentación.

De este modo, las reglas diseñadas para la colección presentaciones son las siguientes: solo puede leer y escribir en ella y en los documentos contenidos en ella un usuario que esté autenticado como profesor y que además, sea el propietario del documento en el que está intentando leer o escribir. De esta forma se evita que un alumno malicioso pueda crear presentaciones. Adicionalmente se evita que un profesor malicioso pueda modificar presentaciones de otro profesor o crear presentaciones en su nombre.

Para la sub-colección calificaciones las reglas que se han diseñado son las siguientes: sólo puede realizar operaciones de escritura el alumno que ha creado el documento calificación y solo puede realizar operaciones de lectura el profesor que ha creado el documento presentación en el cual está contenida la sub-colección calificaciones de la que se está intentando leer (o el documento calificación dentro de dicha sub-colección). Además, sólo se puede escribir en ellas si el campo isFinished de la presentación que la contiene es false, de esta forma se garantiza que una vez acabada la presentación, un alumno malicioso pueda seguir poniendo calificaciones. Estas reglas también garantizan que un profesor malicioso no pueda modificar las notas puestas por sus alumnos y que un profesor malicioso pueda leer las notas de la presentación creada por otro profesor.

Los documentos calificación, al contrario que los documentos presentación, no tienen un campo owner. Esto se debe a que el ID del documento presentación está programado para que sea igual al ID del alumno que lo crea. Los motivos para ello se explican detalladamente en la *sección 3.6*.

### *2.4. Implementación*

La implementación de la base de datos se ha realizado directamente desde la consola de Firebase, desde la cual, al tener el rol de administrador, se puede acceder libremente a todas las colecciones sin tener que ceñirse a las reglas de seguridad.

En primer lugar se han creado las tres colecciones que no pueden ser modificadas (alumnos, profesores y asignaturas).

Para esta prueba de concepto se han creado cuatro alumnos, tres profesores y cinco asignaturas. A cada alumno o profesor se le ha asignado un e-mail y una contraseña mediante el servicio Firebase Auth.

Por cada asignatura sea creado un documento asignatura, es decir, la colección asignaturas contiene cinco documentos asignatura, lo mismo ocurre con los alumnos y profesores.

No ha sido necesario crear desde la consola la colección presentaciones ni la sub-colección, ya que Firebase por defecto, al intentar escribir en una colección que no existe, la crea, por lo que en el momento de intentar escribir la primera presentación se creó automáticamente la colección presentaciones y al igual con la sub-colección calificaciones.

Por otro lado, las reglas de la base de datos han sido programadas manualmente, también desde la consola, pero siendo necesario escribir el código. Este código se debe escribir en un lenguaje personalizado, propio de Firebase, basado en el Common Expression Language (CEL) con declaraciones match y allow que admiten el acceso

condicional. Mediante la declaración match se especifica la ruta de acceso sobre la que se quiere trabajar (la colección a la que se refieren las reglas) y mediante la declaración allow se otorgan los permisos de lectura y escritura.

Firebase permite además la creación de funciones, se ha hecho uso de esta funcionalidad para crear cinco funciones que nos ayuden en la implementación de las reglas definidas en la *sección 2.3.2*.

- **Función isSignedIn()**:Devuelve una variable de tipo booleano, que toma valor true cuando el usuario está autenticado y false cuando no lo está. Esto se consigue comprobando si la variable request.auth toma o no valor null..

- **Función esProfesor()**: Devuelve un booleano que toma valor true cuando el usuario es un profesor y false cuando no lo es. Esto se consigue haciendo un request del id del usuario y, mediante la función nativa exists(), buscando si coincide con el id de alguno de los profesores que están en la base de datos.

- **Función esAlumno()**: Devuelve un booleano que es true cuando el usuario es un alumno y false cuando no lo es. Esto se consigue haciendo un request del id del usuario y, mediante la función nativa exists(), buscando si coincide con el id de alguno de los alumnos que están en la base de datos.

- **Función isOwner()**:Devuelve un booleano que toma valor true si el id del usuario coincide con el campo owner del documento presentación en el que está intentando leer o escribir.

- **Función presentacionData()**:Mediante el uso de la función nativa get(), devuelve una ruta de acceso al documento presentación que contiene la sub-colección a la que se está intentando acceder. Ha sido necesario crear esta función debido a que, si bien mediante el uso de peticiones simples se puede acceder a los campos de la colección a la que estamos intentando dar permisos, no se puede acceder a los campos de colecciones que estén por encima en la jerarquía, por lo tanto al ir a dar permisos a la sub-colección calificaciones no se podía acceder a los campos isFinished y owner de la presentación.

A la hora de implementar los permisos de la operación create(), se ha usado la variable request.resource, que permite hacer referencia al estado futuro de un documento. De esta forma podemos asegurarnos que un profesor malicioso no pueda crear una presentación añadiendo en el campo owner el id de otro profesor.

### *2.5. Testeo*

Firebase provee de una herramienta para verificar el correcto funcionamiento de las reglas de seguridad programadas. Mediante dicha herramienta se pueden simular peticiones de lectura (get()) y de escritura (create(),update(), delete()) realizadas por un usuario del tipo elegido (sin autenticar, autenticado o autenticado como un usuario concreto) en un path concreto.

Se han realizado pruebas para verificar el correcto funcionamiento de las reglas de seguridad implementadas en todas las colecciones. Para las colecciones alumnos,

profesores y asignaturas se ha realizado una misma prueba, ya que las reglas de acceso, y por tanto el código, son idénticas.

En la figura a continuación se muestra un ejemplo del entorno de pruebas en el que se ha trabajado, mostrando una captura de pantalla de una de las muchas simulaciones realizadas para verificar el correcto funcionamiento de las reglas de seguridad. En este caso corresponde a un intento de realizar una operación de lectura get() de un documento calificación contenido en una presentación de la cual no es el propietario.

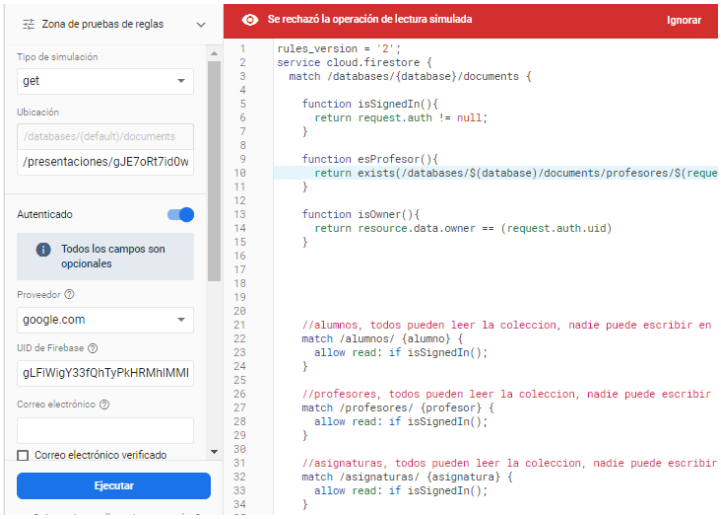


*Figura 2.12. Captura de pantalla de la simulación.*

A continuación se enumeran los test realizados, que cubren todas las posibilidades de intento de acceso a las colecciones por cualquier tipo de usuario:

**a) Colecciones alumnos, profesores y asignaturas:**

1- Usuarios sin autenticar: se ha realizado un test para cada una de las cuatro operaciones posibles get(), create(),delete() o update(). El resultado obtenido es que ningún usuario sin autenticar puede realizar operaciones ni de lectura ni de escritura.

2- Usuarios autenticados: el resultado obtenido es que cualquier usuario autenticado puede leer en las colecciones (operación get()), sin embargo se rechazan los permisos de escritura en las mismas(operaciones create(), update(), delete()).

De este modo, ha quedado verificado el correcto funcionamiento de las normas de seguridad programadas para las colecciones alumnos, profesores y asignaturas.

b) **Colección presentaciones**:

1- Usuarios sin autenticar: el resultado obtenido es que la simulación rechaza todas las operaciones, tanto de lectura como de escritura, para usuarios sin autenticar como era de esperar.

2- Usuarios autenticados como alumnos: el resultado obtenido es que ningún usuario autenticado como alumno puede realizar operaciones de lectura (get()) o escritura (create(), update(), delete()) en la colección presentaciones tal y como estaba previsto.

3- Usuarios autenticados como profesores y que son propietarios del documento presentación: para este caso no se puede probar la operación create() ya que esta es la única operación que no puede referirse a un documento concreto. Así que, para la operación create() no se indica el path a un documento concreto, pero el resultado esperado es que la acepte igualmente ya que esta simulación corresponde a un profesor creando un documento en la colección presentaciones.

   Como era de esperar, se autorizan todas las operaciones cuando el usuario está autenticado como profesor y además es el propietario del documento presentación sobre el que intenta operar.

4- Usuarios autenticados como profesores y que no son propietarios del documento presentación: para este caso no se ha realizado prueba de la operación create() dado que el resultado, y la prueba en sí, serían iguales a las del apartado anterior (un profesor intentando crear un documento en la colección presentaciones). Para el resto de operaciones get(), update() y delete(), se espera que sean rechazadas.

   Los resultados de la simulación coinciden con lo esperado: la simulación rechaza todas las operaciones tanto de lectura como de escritura para usuarios autenticados como profesores que no son propietarios del documento.

   Con estas pruebas, queda comprobado el correcto funcionamiento de las reglas de seguridad programadas para la colección presentaciones.

**c) Sub-colección calificaciones**:

1- Usuarios sin autenticar: el resultado obtenido es que ningún usuario sin autenticar puede realizar operaciones get(), create(),delete() o update() en la sub-colección calificaciones.

2- Usuario autenticado como alumno y que es el que ha creado el documento: El resultado esperado es que acepten las operaciones de escritura (update() y delete()) y que se rechacen las de lectura (get()). Con la operación create() ocurre exactamente lo mismo que en la colección presentaciones, no se indicará el path a un documento en concreto sino a toda la subcolección y se espera que se permita.

   Como era de esperar, se rechazan las operaciones de lectura (get()) y se autorizan las de escritura (create(), update() y delete()).

3- Usuarios autenticados como alumnos que no son el que ha creado el documento: el resultado esperado es que ningún alumno que no sea el que lo ha creado pueda realizar operaciones get(), delete() o update() en un documento de la sub-colección calificaciones. Con la operación create() ocurre lo mismo que en la colección presentaciones, en este caso es redundante.

   El resultado obtenido ha sido el esperado: la simulación rechaza todas las operaciones tanto de lectura como de escritura intentadas por alumnos que no han creado el documento.

4- Usuario autenticado como profesor que ha creado la presentación a la cual pertenece la subcolección calificaciones sobre la que se quiere actuar: el resultado esperado es que se permitan las operaciones de lectura (get()) pero no las de escritura (create(), update(), delete()).

   Los resultados de la simulación coinciden con lo esperado: se permiten las operaciones de lectura (get()) pero se rechazan las de escritura (create(), update(), delete()).

5- Usuario autenticado como profesor que no ha creado la presentación a la cual pertenece la subcolección calificaciones sobre la que se quiere actuar: el resultado esperado es que se rechacen las operaciones tanto de lectura (get()) como las de escritura (create(), update(), delete()).

   Como era de esperar, la simulación rechaza todas las operaciones tanto de lectura como de escritura intentadas por un profesor que no ha creado la presentación a la cual pertenece la sub-colección calificaciones a la que se intenta acceder.

   Mediante estas simulaciones, se comprueba que las reglas de seguridad programadas para la sub-colección calificaciones funcionan correctamente.

# 3. Desarrollo de la aplicación móvil

## 3.1. Definición de requisitos

Los requisitos que debe cumplir la aplicación para funcionar correctamente y que se pueda realizar la co-evaluación de presentaciones son los siguientes:

- La aplicación debe implementar un sistema de autenticación de usuarios para que solo usuarios registrados puedan usarla. La autenticación también es necesaria para que, al comunicarse con Firebase, éste pueda detectar qué usuario en concreto está accediendo, y así aplicar las reglas de seguridad en consecuencia.

- Debe ser capaz de diferenciar si el usuario que intenta acceder, está registrado como profesor o cómo alumno.

- Tiene que poder escribir en la base de datos, para que así los profesores puedan crear documentos presentación y los alumnos documentos calificación.

- Tiene que poder leer en tiempo real de la base de datos, para que los profesores puedan ver en tiempo real las calificaciones que los alumnos otorgan a las presentaciones.

- La aplicación también debe implementar un sistema para garantizar que sólo los alumnos presentes en la presentación puedan escribir calificaciones. Para ello es necesario implementar algún tipo de comunicación entre el profesor y los alumnos.

- Por último, la aplicación debe cumplir todos los requisitos anteriores, siendo además la experiencia de usuario sencilla e intuitiva.

## 3.2. Análisis

A la hora de comenzar a programar la aplicación, la primera decisión que hay que tomar es el sistema operativo para el que se va a programar la aplicación. Como se puede observar en la figura a continuación no existe en el mercado ningún competidor que se acerque mínimamente a las cuotas de mercado que manejan Android (Google) e IOS (Apple) por lo que la decisión estará entre estos dos.

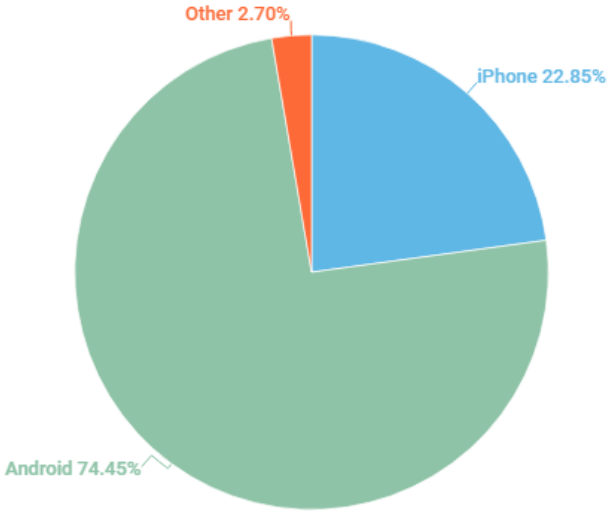


*Figura 3.1. Cuota de mercado Smartphone por sistema operativo.*

Para tomar la decisión entre programar la aplicación para android o IOS, se han tenido en cuenta diversos factores:

- Cuota de mercado: Cuanto mayor sea la cuota de mercado mayor número de usuarios tendrán un Smartphone con ese sistema operativo y podrán usar la aplicación. En este punto y como se puede observar en la *figura 3.1*. Android está considerablemente por delante con una cuota de mercado de 75% frente al 23% de IOS.

- Precio del dispositivo: Los Smartphone Android, son significativamente más baratos que los de Apple. El smartphone más barato que se ha encontrado en la Apple Store cuesta 489 €, mientras que se pueden encontrar Smartphone de gama baja desde 110 €. En el espíritu de intentar llegar a la mayor cantidad de gente antes mencionado, y teniendo en cuenta que la aplicación va dirigida a alumnos, que normalmente no disponen de una gran cantidad de recursos económicos, Android vuelve a estar por delante en este apartado.

- Lenguaje de programación: El lenguaje de programación usado para programar Android es Java, o en su defecto Kotlin, mientras que para IOS el lenguaje que se usa es Swift. Dado que a lo largo de la carrera se ha programado en Java en diversas asignaturas y en ningún caso se ha usado Swift, parece lógico decantarse por Android también en este punto.

- Comunidad: La comunidad de programadores de Android es mucho más cooperativa. Existen numerosas librerías open-source así como páginas y foros que pueden servir de ayuda mientras que la comunidad de IOS es mucho más cerrada.

- Dispositivos necesarios: Debes disponer de un ordenador MAC si quieres programar aplicaciones para IOS mientras que en cualquier PC se pueden programar aplicaciones Android.

Por todas las razones expuestas anteriormente se ha decidido programar la aplicación en Android. Una vez tomada esta decisión, el siguiente paso es decidir en qué entorno de desarrollo se va a programar la aplicación.

Los dos principales entornos de desarrollo para aplicaciones Android son Android Studio y Eclipse. A continuación se describen los factores que se han tomado en consideración para decidir cuál de los dos utilizar:

- Experiencia previa: Durante la carrera, Eclipse ha sido el entorno de desarrollo para programar en Java mientras que usar Android Studio implica aprender a trabajar en un entorno de desarrollo completamente nuevo desde cero.

- Potencia del PC necesaria: Android Studio es un programa mucho más pesado y consume una cantidad de recursos muy superior a Eclipse.

- Entorno oficial: Android Studio es el entorno oficial brindado por Google para el desarrollo nativo en Android, lo que permite una más sencilla inclusión de las librerías y APIs propias de Google que en Eclipse.

- Diseño visual: Android Studio proporciona una herramienta para diseñar las activitys (las pantallas de la aplicación) de forma visual, mientras que en Eclipse algunas de las librerías como CardView o RecyclerView no se reconocen.

- Profesionalidad: Android Studio es la herramienta usada por la mayoría de los desarrolladores de aplicaciones y su tendencia de uso está aumentando entre ellos cada vez más con los años debido al respaldo de Google.

Teniendo en cuenta que en el PC del que se dispone Android Studio funciona perfectamente fluido, la única ventaja que tiene Eclipse es la experiencia previa trabajando con el mismo. Comparando esto contra todas las ventajas que aporta Android Studio explicadas previamente, se ha decidido utilizar Android Studio como entorno de desarrollo de la aplicación.

### *3.3. Diseño*

En cuanto al diseño de la aplicación, en primer lugar se ha diseñado la comunicación entre el profesor y los alumnos, para verificar la asistencia de los mismos a la presentación.

Se ha optado por realizar una comunicación vía Bluetooth Low Energy. El profesor envía vía BLE en modo broadcast una contraseña que los alumnos recogen y necesitan para poder votar.

La decisión de usar la tecnología BLE se debe a que su limitado rango de acción (aproximadamente 10 metros), permite garantizar que el alumno que recibe la contraseña está en la clase.

En cuanto al diseño del código de la aplicación, se han creado tres clases que implementan métodos para:

1- Gestionar la autenticación de ususarios: Clase AppAuthManager.

2- Gestionar las operaciones de lectura y escritura de la base de datos: Clase AppDatabaseManager.

3- Manejar la comunicación Bluetooth: Clase AppBleManager.

Se han programado varias actividades, cada una de ellas con una función concreta:

1- MainActivity: Es la primera actividad que se lanza, es la encargada de comprobar el estado de conexión del usuario que intenta acceder (Si un usuario inicia sesión, se guarda su estado incluso aunque cierre la aplicación). También se encarga de la diferenciación de roles.

2- AuthenticationActivity: Esta actividad se encarga de pedir y comprobar las credenciales de inicio de sesión de los usuarios que intentan acceder.

3- ProfesorActivity: Se encarga de mostrar por pantalla al profesor todas las asignaturas que imparte, para que pueda elegir a que asignatura corresponde la presentación que quiere crear.

4- PresentacionActivity: Es la encargada de crear la presentación en la base de datos.

5- CalificaciónActivity: Esta actividad tiene varias funciones. Por un lado se encarga de las escucha en tiempo real de las calificaciones que escriben los alumnos y además se encarga de la creación y envío del paquete BLE con la contraseña en modo broadcast. También es la encargada de finalizar las presentaciones.

6- AlumnoScanActivity: Encargada de la recepción del paquete BLE y la decodificación de la contraseña.

7- AlumnoActivity: Se encarga de escribir la nota que pone el alumno en la sub-colección calificaciones de la presentación creada.

8- AcabarActivity: Muestra al profesor un resumen de la presentación.

Las diferentes actividades de la aplicación interactúan con las clases antes mencionadas para llevar a cabo las labores que cada una tiene asignada. En la figura a continuación se muestra el diagrama de clases de la aplicación, donde se puede observar qué actividades interactúan con cada una de estas clases. Cabe destacar que AcabarActivity no aparece en el diagrama de clases, ya que simplemente muestra un resumen y no interactúa con ninguna de las clases creadas.

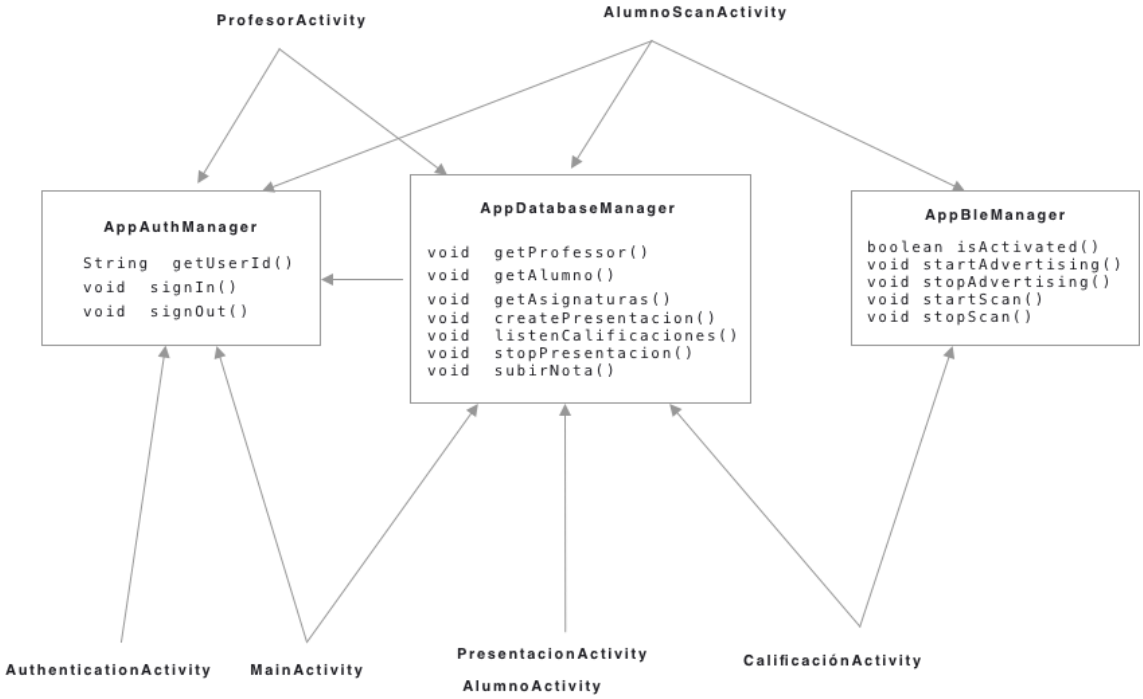


*Figura 3.2. Diagrama de clases aplicación.*

## 3.4. Implementación

En esta sección se explicará la implementación y funcionamiento de la aplicación móvil. Esto incluye el control de la autenticación de usuarios, la diferenciación de los roles profesor y alumno, el uso de la comunicación Bluetooth Low Energy (BLE) para la verificación de la asistencia, la metodología utilizada para la asignación de notas y su subida a la base datos y, por último, la interfaz gráfica y la experiencia de usuario en la navegación por la aplicación.

### 3.4.1. Autenticación de usuarios

Para llevar a cabo la autenticación de usuarios, se ha utilizado el SDK de Firebase. Éste proporciona una forma segura de comunicación entre la aplicación y el servidor. La forma que se ha usado para identificar a los usuarios es el correo electrónico y una contraseña (esto son las credenciales) que son mandadas al servidor. Después, en el servidor, Firebase valida estas credenciales y manda de vuelta un token de autenticación a la aplicación. Este token contiene la información del usuario que acaba de iniciar sesión. La funcionalidad más poderosa que se consigue, es que una vez que un usuario inicia sesión, Firebase mantiene el estado del usuario. Esto quiere decir que aunque el usuario cierre la aplicación, la sesión se mantendrá iniciada.

### 3.4.2. Diferenciación de roles

Una vez el usuario está autenticado, el siguiente paso es distinguir si se trata de un profesor o de un alumno, para ello en primer lugar se recupera una instancia de la base de datos.

A continuación, se realiza una búsqueda en la colección profesores del ID del usuario registrado. Si se encuentra, se guarda el campo nombre y se considera que el usuario

que ha iniciado sesión es un profesor, inicializando *ProfesorActivity*, pasándole como parámetro un String con el nombre del profesor que ha iniciado sesión.

Si no se encuentra el ID del usuario en la colección profesores, se busca de igual manera en la colección alumnos. Para ello es necesario instanciar de nuevo la base de datos. Al igual que con la colección profesores, si se encuentra el ID del usuario en la colección alumnos se guarda el campo nombre, se considera que el usuario que ha iniciado sesión es un alumno y a continuación se lanza *AlumnoScanActivity.java* pasándole como parámetro un String con el nombre del alumno en cuestión.

### *3.4.3. Creación de presentaciones*

Los usuarios con el rol profesor son los encargados de generar los documentos presentación dentro de la colección presentaciones. Una vez un usuario profesor inicia sesión llega a la clase *ProfesorActivity.java* como se explica en la *sección 3.4.2*. El profesor verá por pantalla una lista con todas las asignaturas que imparte y tendrá que elegir cuál de ellas corresponde la presentación que quiere crear.

Para poder mostrar la lista de asignaturas, en primer lugar se ha creado una clase Asignatura.java cuyos atributos son String id, String name, List<String> alumnos y List<String> profesores. Esta clase también contiene un método *getNumberOfAlumnos()* que devuelve el tamaño de la lista de alumnos.

A continuación se realiza una búsqueda en la base de datos de todos los documentos de la colección asignaturas y se busca en el campo "profesores" que el ID del usuario coincida con el valor de alguna de las posiciones del vector (recordar que en cada una de las posiciones del vector de Strings del campo profesores está el ID de cada uno de los profesores que imparten la asignatura).

Se crea una lista de objetos de tipo Asignatura, y por cada coincidencia se añade un objeto a la lista. Los atributos de estos objetos son asignados a partir de los campos de la asignatura que están en la base de datos.

Una vez creada la lista de asignaturas, se realiza un display de las asignaturas. Cuando el usuario pulsa en la asignatura de la cuál desea crear la presentación, se lanza *PresentacionActivity*.

*PresentacionActivity* recibe como parámetros el ID y el nombre de la asignatura. Cabe destacar que simplemente pasando el nombre o el ID, se puede buscar el otro en la base de datos, pero es más eficiente pasarlo como parámetro que hacer otro request a la base de datos.

Una vez en *PresentaciónActivity*, el usuario ve por pantalla un cuadro para introducir el nombre que quiera ponerle a la presentación y un botón para comenzar la presentación.

Para darle un ID aleatorio al documento presentación que se va a crear, se ha programado una clase llamada *AutoId* que contiene un método *autoId(int idSize)* que crea un String, de letras mayúsculas y minúsculas y números, aleatorio, del tamaño de idSize.

Para poder dar valores a los distintos campos del documento presentación, se crea un HashMap. Se añaden al HashMap los campos del documento presentación: nombre asignatura e id_asignatura, nombre presentación, owner y por último isFinished.

El nombre y el ID de la asignatura se recibieron como parámetros al iniciar *PresentacionActivity.java*, el nombre de la presentación es el que escribió el usuario antes de pulsar el botón para comenzar la presentación, el campo owner, que es el ID del profesor que ha iniciado sesión, y el booleano isFinished se pone siempre a falso al crear la presentación, ya que no está acabada.

Se ha elegido un tamaño de 20 bytes para el ID de las presentaciones que se crean. La justificación de esta decisión se explica con detalle en la *sección 3.4.4*.

Una vez generados el HashMap "presentation" y el ID de la presentación "idPresentacion", se crea el documento en la base de datos e inmediatamente después se lanza la clase *CalificacionActivity.java*, la cual recibe como parámetros el ID de la asignatura, el ID de la presentación y el nombre de la presentación.

### *3.4.4. Verificación de asistencia*

Para verificar la asistencia de los alumnos a la presentación que van a co-evaluar, se ha recurrido a la comunicación Bluetooth Low Energy (BLE). El profesor envía vía BLE en modo broadcast una contraseña que los alumnos recogen y necesitan para poder votar.

#### *3.4.4.1. Creación de la contraseña*

Como se explica en el anexo I, se dispone de un máximo de 29 bytes para enviar en el paquete de advertising, por lo que se ha adaptado el tamaño de la contraseña a esta limitación y su longitud es de 26 bytes. El motivo de usar 26 bytes y no los 29 bytes de los que se dispone, es que el payload se usa también para enviar la potencia de transmisión de la señal y esta ocupa 3 bytes. Si se quisiera una contraseña más robusta, se podría optar por omitir el envío de la potencia de transmisión y así poder disponer de una contraseña de una longitud 3 bytes mayor. Sin embargo se ha optado por lo contrario, ya que la contraseña realmente se envía de forma broadcast y no es la longitud de la contraseña lo que aporta seguridad a la aplicación, sino las reglas de seguridad de la base de datos.

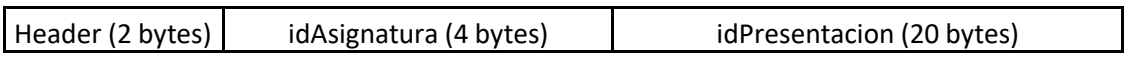


*Figura 3.3. Formato contraseña.*

La contraseña consiste en tres Strings concatenados, el primero una cabecera de dos bytes cuyo valor es "AA", que sirvió durante la fase de testeo del envío y recepción de las tramas de advertising para diferenciar a primera vista el dispositivo que se estaba buscando entre la multitud de otros dispositivos BLE que hay en cualquier lugar actualmente.

El segundo String corresponde al ID de la asignatura a la cual pertenece la presentación que se ha creado. Se incluye el ID de la asignatura para facilitar la experiencia de

usuario de los alumnos, y que automáticamente se conecten a la presentación a la que están asistiendo. Esto se explica más en detalle en la *sección 3.4.5*.

El tercer y último String corresponde al ID de la presentación que se ha creado. Este String es el que actúa como una "contraseña" al uso, ya que para escribir en un documento de Firebase, es necesario conocer su ID. De esta forma se garantiza que si un alumno no está físicamente en la presentación y no recibe el paquete de advertising BLE con el ID de la presentación, nunca podrá calificarla.

Los tamaños de diseño tanto de idAsignatura (4 bytes) como de idPresentacion (20 bytes) se han decidido por una mera cuestión lógica: El número de asignaturas es finito, es decir hay un número fijado de asignaturas en cada centro, que pueden variar de un año a otro, pero no significativamente y con 4 bytes se garantiza un número máximo de representaciones de ID de asignaturas de $2^{8*4}$ = 4.294.967.296. Dado este número tan alto, se representan los ID de las asignaturas como Strings de 4 caracteres, ocupando cada uno 1 bytes. El alfabeto que se usa, que incluye todas las letras tanto mayúsculas (A-Z) como minúsculas (a-z) y los números (0-9), consta de un total de 60 caracteres diferentes, por lo tanto hay un total de $60^4$ = 12.960.000 posibles ID de asignaturas.

Por otro lado, dado que el número de presentaciones creadas será significativamente mayor, se han reservado más bytes para el ID de las mismas. Ya que se usa la misma técnica de representación que para los ID de las asignaturas, se pueden crear un total de $60^{20}$ presentaciones con un ID único.

Para la creación del String concatenado, se ha creado una clase llamada *AdvertisingDataHelper* que contiene dos métodos, uno que concatena dos Strings que se le pasan como parámetro y les añade el header "AA" al comienzo llamado *generateDeviceName(String, String)* y otro llamado *recoverIds(String)* que recibe como parámetro un String de la forma de la contraseña y elimina la cabecera y devuelve un par de Strings que corresponden a idAsignatura e idPresentación. Este método es al que llamarán los alumnos para recuperar el ID de la asignatura y el ID de la presentación.

#### *3.4.4.2. Configuración BLE y envío del paquete de advertising*

La creación y envío de la trama de advertising BLE se realiza en su totalidad desde *CalificacionActivity*. Todo este proceso se lleva a cabo usando las clases y métodos del API de android android.bluetooth y android.bluetooth.le

En primer lugar se inicializa el BluetoothAdapter, que representa el adaptador Bluetooth del propio dispositivo. Existe un adaptador de Bluetooth para todo el sistema y la aplicación actúa con él usando este objeto. Este es el primer paso para realizar cualquier tipo de acción Bluetooth, ya sea hacer advertising, escanear dispositivos o cualquier otro tipo de acción relacionada con Bluetooth y no tanto con BLE como crear un BluetoothServerSocket para escuchar peticiones de conexión RFComm o L2CAP.

A continuación, se asegura que el dispositivo tenga el Bluetooth activado y si no lo está se pide por pantalla que se active.

Una vez el Bluetooth está activado, comienza el advertising.

Se ha decidido enviar la contraseña en el nombre del dispositivo, ya que los bytes ocupados por éste ya están incluidos en el payload. De esta forma se ha ahorrado espacio para poder hacer la contraseña de 26 bytes, ya que de lo contrario habría que, o bien reducir el tamaño de la contraseña tanto como fuera el nombre del dispositivo, o bien no enviar el nombre del dispositivo en el paquete de advertising y enviar la contraseña en otra parte del payload, como por ejemplo los datos de servicio que actualmente no se mandan.

Sabido esto, el siguiente paso es crear el String contraseña mediante el método *generateDeviceName()* de la clase *AdvertisingDataHelper* antes mencionada, pasándole como parámetros idAsignatura e idPresentacion. Una vez creado este String, se asigna como nombre del dispositivo.

A continuación se crea el advertiser, que es el equivalente al adaptador bluetooth, que nos permite hacer todas las operaciones relacionadas con el advertising.

Una vez creado el advertiser, se le asignan valores a sus parámetros:

- *setConnectable()*: da la opción de permitir a otros dispositivos establecer una conexión, por lo que se pone false.
- *setScannable():* permite que otros dispositivos que estén escaneando detecten el paquete de advertising por lo que se pone true.
- *setInterval():* determina la frecuencia con la que se mandan los paquetes de advertising, al usar el flag INTERVAL_HIGH se selecciona la frecuencia más baja posible, mandándose un paquete cada aproximadamente 1000ms.
- *setTxPowerLevel():* determina la potencia de la transmisión, aunque esta viene determinada también por el dispositivo específico que está enviando los paquetes.

Finalmente, una vez preparado el paquete de advertising, comienza su envío broadcast. El advertising BLE utiliza un callback para devolver el estado de la operación de advertising.

#### *3.4.4.3. Recepción del paquete de advertising*

La recepción del paquete de advertising y la decodificación de la contraseña se realiza en la clase AlumnoScanActivity.java.

En primer lugar, al igual que para el envío del paquete de advertising, se incializa bluetoothAdapter y se asegura que el dispositivo tenga encendido el Bluetooth (*ver sección 3.4.4.2).*

A continuación, de manera análoga a cuando se hace el display de las asignaturas del profesor (ver *sección 3.4.3*), se crea una lista con objetos asignatura que contiene todas las asignaturas de la base de datos en las que el ID del usuario aparece en alguna posición del array del campo alumnos. Una vez creada la lista de asignaturas, se hace visible un botón para empezar el escaneo de dispositivos BLE. Esto se hace así por seguridad, para evitar que si hay un fallo en la descarga de la lista de asignaturas, un alumno empiece a escanear.

Cuando el usuario pulsa el botón para empezar a escanear, comienza el escaneo. Se ha implementado una funcionalidad para que el escaneo pare automáticamente si tras 10 segundos no ha encontrado ningún dispositivo.

Los resultados del escaneo se guardan en scanCallback, que es un objeto de la clase abstracta *ScanCallback*. Esta clase contiene el método *onScanResult()* que permite acceder al registro de escaneo y se ejecuta cada vez que se encuentra un paquete de advertising BLE.

Para poder acceder al nombre del dispositivo, hay que convertir el objeto result de la clase *ScanResult* a un objeto de la clase *ScanRecord*, ya que la clase ScanResult no implementa ningún método para conseguir el nombre del dispositivo. Para conseguir un objeto de la clase *ScanRecord* se usa el método *getScanRecord()* implementado en la clase *ScanResult*.

A continuación, se usa el método *getDeviceName()* de la clase *ScanRecord* para recuperar el nombre del dispositivo, es decir, el String concatenado que el profesor está emitiendo. Una vez recuperado el nombre del dispositivo, se usa el método *recoverIds()* de la clase *AdvertisingDataHelper* para eliminar la cabecera y separar el ID de la asignatura y el ID de la presentación.

Por último, se busca en la lista de asignaturas si el ID de la asignatura extraído coincide con alguno de la lista de asignaturas creada. Dado que para esta prueba de concepto, se ha supuesto que un alumno no puede tener que atender a dos presentaciones al mismo tiempo, el hecho de que el ID de la asignatura extraído coincida con alguno de la lista garantiza que ese es el dispositivo del profesor que está emitiendo por lo que se para el escaneo y se llama a la clase *AlumnoActivity* pasándole como parámetros el nombre del alumno y el ID de la presentación que se acaba de decodificar.

Por el contrario, si ID asignatura decodificado no coincide con ninguno de la lista o bien en cualquiera de los pasos anteriores el resultado es null, el dispositivo no será el del profesor que ha creado la presentación que el alumno está observando (puede ser un dispositivo cualquiera o incluso otro profesor que está usando la aplicación en otra clase de otra asignatura distinta, por lo que es importante recalcar que para esta prueba de concepto se ha supuesto que un alumno no puede estar en la lista de dos asignaturas que se impartan al mismo tiempo), y por lo tanto se descartará y el escaneo seguirá con normalidad hasta que se encuentre un dispositivo válido o pasen 10 segundos sin encontrar ningún nuevo dispositivo.

### *3.4.5. Asignación de notas*

Los usuarios autenticados como alumnos que sepan el ID de una presentación activa, pueden escribir documentos calificación en la sub-colección calificaciones de esa presentación, acorde con lo programado en las reglas de seguridad de la base de datos. Mientras tanto, el profesor escucha en tiempo real de la base de datos para comprobar las notas que se van subiendo.

#### 3.4.5.1. Creación de los documentos calificación en la base de datos

La asignación y posterior subida de la nota a la base de datos se lleva a cabo desde la clase *AlumnoActivity.java*: esta clase recibe de *AlumnoScanActivity.java* el ID de la presentación sobre la que se va a escribir y el nombre del alumno que ha iniciado sesión.

El usuario alumno, ve por pantalla un cuadro de texto donde se le pide que introduzca la nota y un botón para subirla. Al pulsar el botón, se crea un documento calificación en la sub-colección calificaciones de la presentación con el ID que se ha recibido en el paquete BLE.

La técnica utilizada para crear documentos calificación es la misma que la usada para crear presentaciones (ver *sección 3.4.3*). La diferencia es que, en este caso, los campos que se crean en el HashMap son únicamente nombre_alumno que corresponde al nombre del alumno que está calificando la presentación y calificacion, que corresponde a la nota que ha escrito el alumno en el cuadro de texto.

Una vez creado el HashMap, se puede crear el documento calificación. Para ello hay que especificar la ruta completa, por eso es indispensable conocer el ID de la presentación para poder escribir una calificación.

Para evitar que un alumno malicioso intente escribir más de una calificación, pero a la vez permitirle rectificar si se ha equivocado al poner la calificación, se aprovecha que Firebase, al intentar escribir un documento con el ID de un documento ya existente, automáticamente lo reemplaza. De esta forma, en lugar de usar el método *autoId()* para asignar un ID aleatorio al documento calificación que se va a crear, se fuerza a que el ID del documento calificación sea igual al ID del alumno que lo está creando.

Una vez que el documento calificación es creado correctamente, se muestra un pop up que avisa de que la calificación ha sido enviada con éxito y se llama de nuevo a *AlumnoScanActivity*, para poder escanear en busca de una nueva presentación.

#### 3.4.5.2. Escucha en tiempo real de las calificaciones

La escucha en tiempo real de las calificaciones por parte del profesor se realiza desde *CalificacionActivity*. Se leen las calificaciones en tiempo real y se muestran por pantalla.

Para mostrar por pantalla las calificaciones, en primer lugar se ha creado una clase Calificacion.java para poder crear objetos calificación. Los objetos calificación tienen los atributos nombreAlumno y nota.

- nombreAlumno: corresponde al nombre del alumno que ha creado el documento calificación.
- nota: corresponde a la calificación que el alumno le ha dado.

Este procedimiento es análogo al mostrado en la *sección 3.4.3* para la creación de la lista de asignaturas que imparte el profesor.

Para realizar la escucha, en primer lugar se lleva a cabo una consulta simple de la colección de la que se va a escuchar.

A continuación, el método *addSnapshotListener()* permite registrar un objeto que implemente la interfaz *ListenerRegistration*. Este objeto será notificado por el stack de Firebase cada vez que haya una actualización en la colección a la que apunta la consulta previamente creada.

Para cada actualización que haya en la base de datos, se crea un objeto de tipo calificacion y se añade a la lista de calificaciones. Inmediatamente después se llama al método *displayData()* que es el encargado de mostrar la lista de calificaciones por pantalla.

### 3.4.6. Finalización de la presentación

La finalización de la presentación se hace también desde *CalificacionActivity*. El usuario profesor ve por pantalla un botón para darla por finalizada cuando todos los alumnos hayan enviado su calificación.

Pulsar este botón muestra un pop up para asegurar que el profesor quiere acabar la presentación, y en caso afirmativo se llama al método *acabarPresentacion()*.

El método *acabarPresentacion()*, realiza tres tareas para dar por finalizada la presentación.

1- Elimina el listener para dejar de realizar escuchas de las actualizaciones de la base de datos.

2- Interrumpe el envío de paquetes advertising BLE con la contraseña.

3- Modifica el campo isFinished de la presentación en la base de datos y lo pone a true, de esta forma no se podrá escribir ningún documento calificacion más dadas las reglas de seguridad que se han programado en la base de datos.

Una vez realizadas estas tres tareas, se llama a la clase *AcabarActivity.java*, pasándole como parámetros el nombre de la presentación y la media de todas las calificaciones y se cierra *CalificacionActivity.java*.

*AcabarActivity.java* es una clase que simplemente muestra por pantalla el nombre de la presentación y la media de las calificaciones y tiene un botón para salir definitivamente la presentación y volver a *ProfesorActivity.java*.

### 3.4.7. Interfaz gráfica y navegación

En Android, cada actividad lleva asociado un layout, que es lo que el usuario ve por la pantalla del dispositivo. Al igual que se programa el código para modelar el comportamiento de la aplicación hay que escribir código para crear el layout y luego desplegarlo al comienzo del método *onCreate()*.

El método *onCreate()* se ejecuta inmediatamente cuando se llama a la actividad, y por lo tanto contiene el código que modela el comportamiento de la aplicación.

El layout, es un archivo .xml que mediante el método *setContentView()* se despliega y especifica lo que se muestra por pantalla cuando se inicia la actividad.

#### 3.4.7.1. Pila de actividades Android

Las diferentes actividades de una aplicación Android se organizan en la llamada pila de actividades y se colocan en el orden en que se abre cada actividad. La actividad que esté más arriba en la pila es la que tiene el foco, es decir, es la que se está ejecutando y por tanto la que genera lo que el usuario ve por pantalla. Si una actividad llama a otra y no es cerrada específicamente mediante el método *finish()*, se quedará en la pila, por debajo de la actividad que se acaba de abrir, pero se conservará su estado (como la posición de deslizamiento y el texto ingresado en los formularios). Si el usuario pulsa el botón Back, se elimina de la pila la actividad actual y se reanuda la actividad que estaba inmediatamente por debajo en la pila y se restablece su estado.


*Figura 3.4. Pila de actividades Android.*

#### 3.4.7.2. Navegación de un usuario profesor

Cuando un usuario cualquiera entra a la aplicación, lo primero que se ejecuta es *MainActivity*. Esta actividad detecta si el usuario había iniciado sesión previamente, si no lo había hecho, se llama a *AuthenticationActivity*, que muestra por pantalla dos cuadros de texto en los que el usuario debe introducir su correo electrónico y su contraseña. *MainActivity* se finaliza, y se elimina de la pila de aplicaciones.


*Figura 3.5. Captura de pantalla AuthenticationActivity.*

Una vez iniciada sesión, AuthenticationActivity.java se finaliza y se elimina de la pila de aplicaciones y el usuario profesor es enviado a *ProfesorActivity*, donde se muestra su nombre en la parte superior de la pantalla, así como una lista con todas las asignaturas que imparte y el número de alumnos matriculados en cada asignatura.

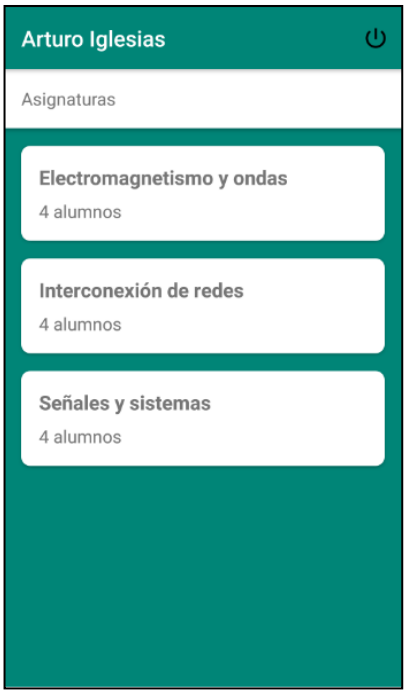


*Figura 3.6. Captura de pantalla ProfesorActivity.*

Adicionalmente, hay un botón en la parte superior derecha de la pantalla para cerrar sesión, al pulsarlo, se muestra un pop-up para asegurar que se quiere cerrar la sesión y si se confirma, se vuelve a llamar a *AuthenticationActivity*.

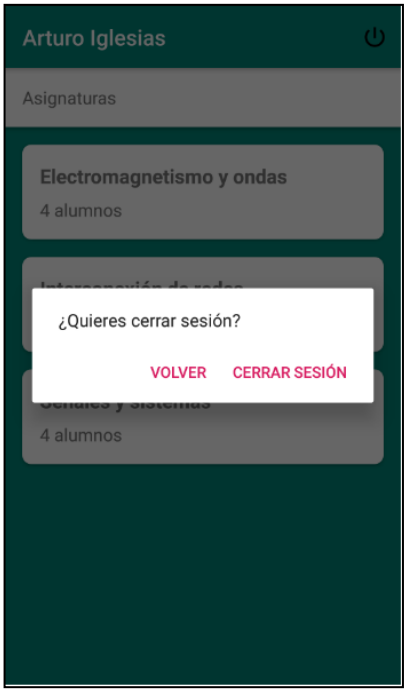


*Figura 3.7. Captura de pantalla ProfesorActivity pop-up.*

Cuando el usuario profesor pulsa una de las asignaturas de su lista, esto indica que quiere crear una presentación de dicha asignatura y se llama a *PresentacionActivity*, donde se muestra por pantalla en la parte superior la asignatura de la cuál va a ser la presentación. También se muestra un cuadro de texto para que se introduzca el nombre de la presentación que se quiere crear. Por último, incluye un botón para dar comienzo a la presentación. *ProfesorActivity* no se finaliza, y se mantiene en la pila de aplicaciones.


*Figura 3.8. Captura de pantalla PresentacionActivity.*

Una vez el usuario profesor presiona el botón para empezar la presentación, se lanza *CalificacionActivity*, donde se muestra el nombre de la presentación y las notas que los alumnos van subiendo a la base de datos en tiempo real. Además, se dispone de un botón para dar por finalizada la presentación en la parte inferior de la pantalla. Al lanzar

*CalificacionActivity*, *PresentacionActivity*. se finaliza, y se elimina de la pila de aplicaciones.

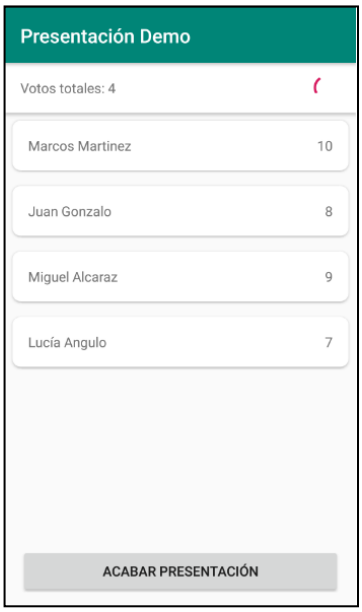


*Figura 3.9. Captura de pantalla CalificacionActivity.*

Si el usuario profesor pulsa el botón Acabar presentación, se muestra un pop-up para asegurar que no se cierra la presentación por error.

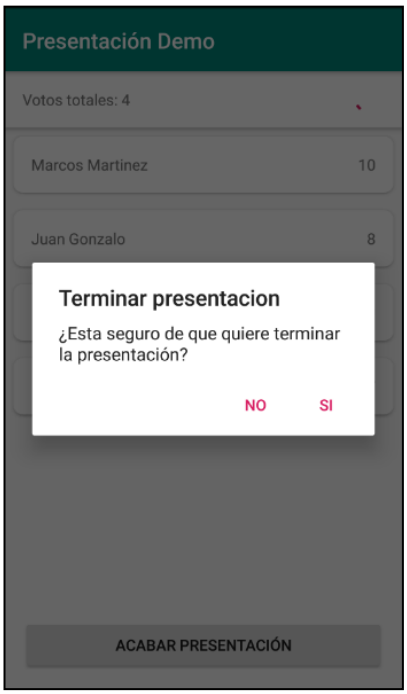


*Figura 3.10. Captura de pantalla CalificacionActivity pop-up.*

Al confirmar que se quiere cerrar la presentación, se lanza *AcabarActivity*, donde se muestra por pantalla el nombre de la presentación y la media de las notas que han puesto los alumnos. También hay un botón para salir de la presentación actual. Al lanzar *AcabarActivity*, *CalificacionActivity* se finaliza y se elimina de la pila de aplicaciones.


*Figura 3.11. Captura de pantalla AcabarActivity.*

Pulsar el botón Salir, finaliza *AcabarActivity*, y por lo tanto se volverá a la aplicación que esté más arriba en la pila de aplicaciones, que es *ProfesorActivity*, por lo que el usuario profesor verá automáticamente por pantalla la lista de las asignaturas que imparte y estará listo para iniciar otra presentación.

#### 3.4.7.3. Navegación de un usuario alumno

Cuando un usuario inicia sesión como alumno, *AuthenticationActivity* llama a *AlumnoScanActivity*. Esta actividad muestra por pantalla (una vez que se ha descargado de la base de datos la lista de asignaturas del usuario alumno) un botón para empezar el scan de dispositivos Bluetooth. Una vez pulsado este botón, cuando se encuentra el dispositivo BLE del profesor, se llama a *AlumnoActivity*. *AlumnoScanActivity* no se finaliza, y permanece en la pila de actividades.

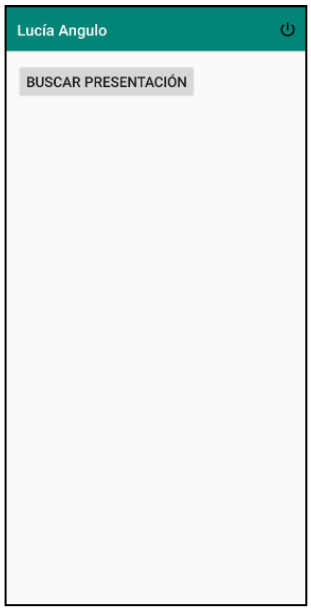


*Figura 3.12. Captura de pantalla AlumnoScanActivity.*

En esta actividad, el alumno ve por pantalla un cuadro de texto para introducir la calificación que cree correspondiente y un botón para subirla a la base de datos. Una vez se pulsa el botón, se crea el documento calificaciones en la base de datos y cuando esto ocurre aparece un pop-up avisando de que la calificación se ha enviado con éxito y con un botón para terminar. Pulsar este botón cierra *AlumnoActivity*, regresando a la siguiente actividad de la pila que es *AlumnoScanActivity*.

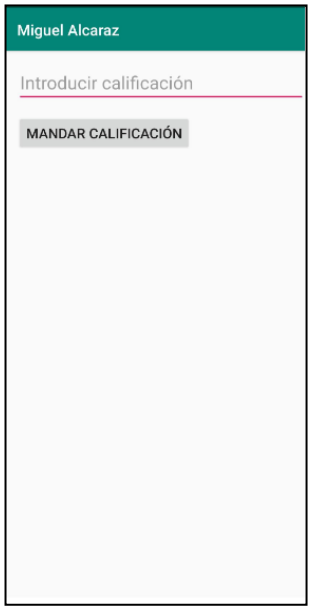


*Figura 3.13. Captura de pantalla AlumnoActivity.*

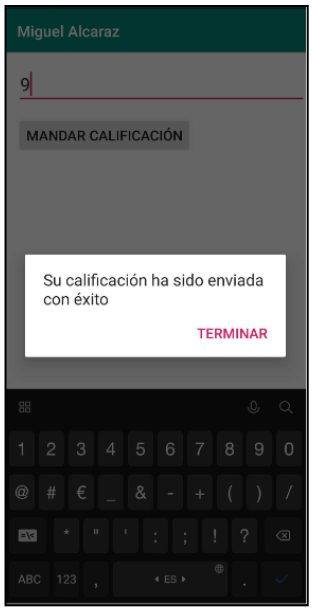


*Figura 3.14. Captura de pantalla AlumnoActivity pop-up.*

## 3.5. Testeo

Para el testeo de la aplicación se han aplicado diferentes métodos atendiendo a qué parte de la misma se iba a testear: Testeo de la codificación y decodificación del paquete de advertising BLE con la contraseña, testeo de la comunicación bluetooth, testeo de la escucha en tiempo real de las calificaciones y testeo de la generación de documentos en la base de datos.

1- Testeo de la contraseña: Para el testeo de la codificación y decodificación de la contraseña, que se realiza mediante los métodos de la clase *AdvertisingDataHelper* (como se explica en la *sección 3.4.4.1*), se ha llevado a cabo un Unit Test en Andorid Studio. Esta herramienta permite testear que la concatenación y posterior desconcatenación de los Strings cabecera, idAsignatura e idPresentación se realiza de manera correcta. Esto se consigue ya que Android Studio compara el resultado que devuelven los métodos *generateDeviceName()* y *recoverIds()* con un resultado esperado que se introduce manualmente.

   De esta forma se han creado los tres Strings y se han concatenado usando el método *generateDeviceName()* y de forma manual y el resultado obtenido ha sido que este método funciona correctamente, se concatenan los tres Strings.

   De forma análoga se ha realizado un test usando como entrada el String obtenido como resultado en el test anterior para verificar el correcto funcionamiento del método *recoverIds()*. El resultado ha sido el esperado y este método devuelve una variable de tipo Pair con el ID de la asignatura (es decir, los caracteres del 3 al 6 del String concatenado) en el primer String del par y el ID de la presentación (los caracteres del 7 al 26) en el segundo String del par.

   La realización de estos dos test garantiza que la contraseña se codifica y se decodifica de forma correcta mediante los métodos creados en la clase *AdvertisingDataHelper*.

2- Testeo de la comunicación BLE: Para el testeo de la comunicación BLE, se ha utilizado una aplicación comercial llamada nRF Connect que permite detectar dispositivos BLE, mostrando el nombre de los dispositivos que encuentra. De esta forma se comprobó que se estaban enviando las tramas correctamente.

3- Testeo de la escucha en tiempo real: El testeo de la escucha en tiempo real se ha realizado usando 5 smartphones simultáneamente. Uno de ellos ha iniciado sesión como profesor y los otros cuatros como alumnos. Se ha hecho a los alumnos enviar las calificaciones simultáneamente y se ha comprobado in situ desde el móvil del profesor que las calificaciones aparecen en la pantalla segundos después de que los alumnos pulsen el botón de enviar calificación. También se ha comprobado que cuando un alumno modifica su calificación, esta se modifica en el teléfono del profesor al instante.

4- Testeo de la generación de documentos: La correcta generación de los documentos presentaciones y calificaciones en la base de datos se ha corroborado mirando en la consola de Firebase que efectivamente cuándo un profesor pulsa el botón para crear una presentación, esta nueva presentación se puede visualizar en la consola. El mismo método se ha seguido para las calificaciones creadas por los alumnos.

# 4. Conclusiones y líneas futuras

El objetivo fundamental de este trabajo, que era programar una aplicación Android para llevar a cabo la co-evaluación de presentaciones se ha cumplido de forma satisfactoria. Se ha creado desde cero una aplicación funcional que distingue los dos roles necesarios para su funcionamiento (alumno y profesor) que se diferencian mediante el sistema de autenticación de usuarios que proporciona el SDK de Firebase.

Esta aplicación permite a los usuarios con el rol profesor crear documentos presentación en la base de datos y a los alumnos crear documentos calificación en las propias presentaciones para llevar a cabo la co-evaluación, todo esto apoyado en las normas de seguridad de la base de datos que la protegen de que se creen usuarios ficticios (prohibiendo el acceso de escritura en las colecciones alumnos, profesores y asignaturas) o de que un usuario malicioso ya registrado, intente realizar acciones que no corresponden a su rol (limitando los accesos de lectura o escritura a la colección presentaciones y a la sub-colección calificaciones según lo explicado en la *sección 2.4.1*.

Además, se ha implementado un sistema que, mediante el envío de una contraseña en los paquetes de advertising BLE, verifica la presencia de los alumnos en la presentación, evitando que alumnos maliciosos puedan co-evaluar a sus compañeros sin estar siquiera presentes en la presentación.

Sin embargo, esta aplicación, al estar desarrollada como una prueba de concepto, aún tiene mucho margen de mejora.

La mejora más significativa que se podría llevar a cabo consistiría en realizar un volcado de la base de datos de la universidad, obteniendo una lista de alumnos, profesores y asignaturas real y mucho más extensa. Unido a esto podría implementarse un mecanismo, por el cual cada vez que se registrara un nuevo alumno o profesor en la base de datos de la universidad, automáticamente se volcara a Firebase y se le creara un usuario. Esto se podría llevar a cabo creando un nuevo rol en la aplicación llamado administrador que fuera el encargado de volcar la base de datos de la universidad y crear los usuarios en Firebase.

Otra mejora puede estar en el diseño de la base de datos, podrían añadirse sub-colecciones dentro de la colección asignaturas para indicar los grupos de mañanas y tardes si los hubiera, a efectos prácticos esas sub-colecciones podrían tratarse como si fueran una colección asignatura individual pero a la hora de configurar las reglas de seguridad de la base de datos requeriría un gran esfuerzo ya que no se podrían poner las mismas reglas para colecciones asignatura con diferentes turnos y para asignaturas con un único turno.

En cuanto al apartado de la verificación de asistencia, se podría añadir seguridad en las reglas de la base de datos para evitar que, un alumno malicioso, que consiga la contraseña aún no siendo alumno de la asignatura, pueda votar en presentaciones de dicha asignatura. Durante la realización del trabajo este ha sido considerado un problema de menor importancia, ya que queda registrado en la base de datos quién es el

alumno que ha votado, por lo que aunque no se impida el acto tramposo, se sabe quién ha sido el infractor.

También se podría mejorar el que, aunque se ha supuesto que un alumno no puede tener que asistir a dos presentaciones a la vez, podría darse el caso de que sí (por ejemplo un alumno repetidor al que se le solapan las clases de dos asignaturas), para solventar este problema habría que modificar el código de la aplicación para que, en lugar de conectarse directamente a la primera presentación de la cuál sea alumno, se le mostraran por pantalla una lista de todas las presentaciones disponibles.

Otra mejora que se podría llevar a cabo está relacionada con que, obviamente, la aplicación necesita conexión a internet para realizar las consultas y las subidas de información a la base de datos. Se podría programar un pop-up, similar al que se ha hecho para recordar encender el bluetooth, que recordara que es necesaria una conexión a internet.

Por último, mencionar que programar la aplicación para dispositivos IOS puede ser una tarea muy interesante en un futuro.

En resumen, esta aplicación puede tener una funcionalidad inmediata siempre que un profesor se tome la molestia de introducir las listas de profesores, alumnos y asignaturas en Firebase, no obstante la misma aún tiene margen de mejora.

Desde un punto de vista personal, éste trabajo me ha ayudado a mejorar mis conocimientos acerca de la programación Android, así como a ser capaz de usar la consola de Firebase para trabajar con una base de datos, cosa que nunca había hecho. Además, me ha aportado un gran entendimiento de cómo funciona tecnología BLE que actualmente está muy de moda y me puede aportar grandes perspectivas laborales.